PARALLEL DSMC SOLUTION OF THREE-DIMENSIONAL FLOW OVER A FINITE FLAT PLATE

Robert P. Nance*

North Carolina State University, Raleigh, North Carolina

Richard G. Wilmoth†

NASA Langley Research Center, Hampton, Virginia

Bongki Moon‡

University of Maryland, College Park, Maryland

H. A. Hassan§

North Carolina State University, Raleigh, North Carolina

Joel Saltz¶

University of Maryland, College Park, Maryland

Abstract

This paper describes a parallel implementation of the direct simulation Monte Carlo (DSMC) method. Runtime library support is used for scheduling and execution of communication between nodes, and domain decomposition is performed dynamically to maintain a good load balance. Performance tests are conducted using the code to evaluate various remapping and remapping-interval policies, and it is shown that a one-dimensional chain-partitioning method works best for the problems considered. The parallel code is then used to simulate the Mach 20 nitrogen flow over a finite-thickness flat plate. It is shown that the parallel algorithm produces results which compare well with experimental data. Moreover, it yields significantly faster execution times than the scalar code, as well as very good load-balance characteristics.

Nomenclature

M_∞	Freestream Mach number
Re	Reynolds number
p_0	Stagnation pressure, bars
p_w	Surface pressure, Pa
q_w	Surface heat flux, W/m^2
t	Time required to compute 1 time step
T_0	Stagnation temperature, K
T_w	Surface temperature, K
$W(n)$	System degradation function
Z_r	Rotational relaxation number

Introduction

The direct simulation Monte Carlo (DSMC) method of Bird[1] has become the standard method for the analysis of hypersonic rarefied flows. Since its inception, the method has been applied to more and more complex configurations, including the Space Shuttle orbiter geometry[2] and the Upper Atmosphere Research Satellite.[3] Furthermore, many DSMC analyses carried out today include physical phenomena such as thermal and chemical nonequilibrium.[4] The combination of complicated geometries and complicated flow physics leads to large processor-time and storage requirements, even for low-density calculations. For near-continuum DSMC applications, the resource requirements can ren-

* Research Assistant, Mechanical and Aerospace Engineering, Student Member AIAA.

† Research Scientist, Aerothermodynamics Branch, Gas Dynamics Division, Senior Member AIAA.

‡ Research Assistant, Computer Science Department.

§ Professor, Mechanical and Aerospace Engineering, Associate Fellow AIAA.

¶ Associate Professor, Computer Science Department.

der a meaningful simulation infeasible on current scalar architectures.

A new computational resource which may be brought to bear on DSMC problems is found in the advent of parallel computing. While parallel programming is still in its formative stages, parallel architectures show promise in being able to complete tasks in a fraction of the time required by contemporary scalar machines. This new architecture thus represents an opportunity to simulate flows at higher densities, or to perform many simulations in the time previously required for one simulation.

A considerable amount of effort has already been put into the parallelization of DSMC algorithms.[5] One of the aims of the present work is to utilize one such parallel implementation to analyze a problem of practical interest. The problem considered is described below.

CNRS Experiment

Figure 1 shows the pertinent dimensions of a finite-thickness flat plate with truncated leading edge tested at zero angle of attack in the SR3 low-density nitrogen tunnel at the Centre National de la Recherche Scientific (CNRS), Meudon, France.[6] The experimental results for the surface heat-transfer rate were compared to Navier-Stokes and DSMC results by the CNRS researchers; the CFD results were shown to match the test data quite well, while the DSMC results overpredicted the heat flux across the length of the plate. Further efforts to correct the discrepancy in the DSMC results made little difference, as shown in the paper by Hash *et al.*[7] This further study included the consideration of effects such as collision model, grid refinement, and nonuniformities in the upstream test section. However, one possible physical phenomenon not considered in any of these solutions was the possibility of a three-dimensional relief effect, which could lower the heat flux to the plate. Therefore, one motivation for pursuing the problem is to obtain a solution for the complete flowfield. To this end, the flat plate was modeled first with the modified F3 code of Rault[8] and then with a modified version of the DSMC3 code of Bird.[9]

The results from these computations suggested that, while no relief effect appeared to be present, increases in grid resolution tended to increase the agreement between the computed and experimental data. Both of these simulations were conducted on scalar machines; it was reasoned that solution on a parallel architecture would allow a simulation with greater resolution to be pursued with considerably less turnaround time.

Flow Conditions and Flow Physics

Table 1 lists the flow conditions for the CNRS experiment. Note that the stagnation temperature of 1100 K is quite low; therefore, vibrational excitation and dissociation are not expected to take place, and the only species considered was N_2. The Variable Hard Sphere (VHS) model of Bird[1] was utilized with a viscosity-temperature exponent of 0.75. Energy exchange between translational and rotational modes was determined through use of the Larsen-Borgnakke method[10] and a rotational relaxation number $Z_r = 5$. The surface of the plate was assumed to be diffusely reflective with full thermal accommodation. It should also be noted that the test conditions correspond to a freestream Knudsen number based on plate length of about 5.5×10^{-3}. Since rarefied-flow conditions are typically assumed to prevail above Knudsen numbers of around 0.01, these conditions correspond to a near-continuum flow.

Note in Figure 1 that the plate is partitioned into four equal sections. In the DSMC results to be discussed later, only one of these portions is considered. This simplification may be made because of the fact that the plate possesses two planes of symmetry, and that the experiment was carried out at zero angle of attack. Thus, consideration of the entire plate is unnecessary.

One of the objectives of this work was to develop a scalable parallel method while leaving the physics modeled by the original scalar code intact. To this end, no modifications have been made to the physical models employed by the original code for the sake of parallelization. Additionally, the possibilities explored in the earlier work on this problem (effect of collision model, upstream flow nonuniformity, *etc*.) have not been considered here, since the earlier studies showed that these variations made little difference in the results.

Parallel Algorithm

The method of parallelization used here utilizes runtime library support to carry out communication and data structure manipulations associated with molecule lists, as well as provide routines for remapping. This library--the CHAOS library--was developed at the University of Maryland, and is based upon the PARTI library developed at the Institute for Computer Applications in Science and Engineering (ICASE) at NASA Langley Research Center.[11,12] The PARTI library was originally built for use with static irregular problems. These are problems where the data access patterns are known only at runtime, but the data access pattern is invariant once it has been defined. The data access patterns in irregular problems are determined through *indirection arrays*, which are arrays whose elements point to elements in another array, as shown in the FORTRAN code fragment below.

```
        do 10 i=1,n
             sum=sum+x(icg(i))
 10     continue
```

The `icg` array in the above is an example of an indirection array.

DSMC represents a dynamic (or adaptive) irregular problem: Data access patterns are known only at runtime, and can change as execution progresses. The data access patterns change because molecules move from cell to cell during the simulation, and molecule information is frequently referenced with respect to the cell in which a molecule resides. CHAOS was developed with such problems in mind, and utilizes a series of preprocessing steps in order to facilitate efficient computation.

First of all, CHAOS determines how data arrays are to be partitioned. This step involves the generation of a *translation table* which maps elements of the data arrays to their owner processors. This table is globally accessible; in this application, the table is replicated on each processor. The second step is the actual remapping of the data; this remapping is carried out through (1) generation of an optimized interprocessor communication schedule and (2) use of scatter-append type procedures to move the data to the appropriate locations. These entities are discussed below.

Since molecules may move from cells owned by one processor to cells owned by another, it is necessary to communicate molecule-based data between processors even if the problem partition does not change. Since such communication is required every time step, communication optimization is crucial for efficient parallel computation in DSMC. Therefore, CHAOS utilizes optimized communication schedules. This optimization is characterized by communication vectorization. In this process, an effort is made to send only a very few large messages instead of many short messages, since long messages are less expensive to communicate. Communication costs are further reduced through the use of lightweight communication schedules. These lightweight schedules are a further enhancement which can be realized because of the data independence characteristic of DSMC; these schedules are considerably cheaper than standard CHAOS schedules because it is not necessary to specify the placement order of cells being transmitted to another processor.

The scatter-append operations are very useful in DSMC because, once the movement phase is completed, DSMC cells can be operated on in any order (unlike CFD, where knowledge of the cell order and an orderly sweep through the domain are very important). Thus, data need only be appended to existing data lists for each processor, and costly reordering of the data is unnecessary.

As discussed earlier, DSMC is a highly dynamic method; that is, the molecules simulated by the code are not uniformly distributed, and the distribution varies considerably as the simulation progresses. To help maintain a good load balance, CHAOS also supports several methods for repartitioning the domain. The basic premise of any load-balancing algorithm is to partition the domain so that each processor must perform approximately the same amount of work. However, a criterion must be selected as the basis for measuring the amount of work owned by each processor. Two possible candidates are the number of molecules in each cell and the compute time required for each cell; these two alternatives will be examined later. In either case, several options are available for decomposing the domain; three possibilities investigated here are recursive coordinate bisection (RCB),[13] recursive inertial bisection (RIB),[14] and one-dimensional chain partitioning.[12]

In the first two algorithms, the domain is recursively halved (with each new portion of the domain possessing an equal amount of “work”) until there are as many subdomains as processors. The difference between RIB and RCB is that RIB chooses the partitioning direction as the direction with the minimum “inertia.” In other words, if the data in the domain tend to be clustered around an axis different from one of the coordinate axes, RIB will find that axis and partition normal to it. On the other hand, RCB simply chooses the coordinate following the largest dimension of the domain (typically the x axis). The third choice--chain partitioning--is a very inexpensive method that works well for certain problems. Here, the domain is partitioned into many contiguous strips, or chains, with each of these chains containing about the same amount of work. The chain-partitioning method implemented here seeks to reduce the remapping cost even further by considering only the cost of computation in determining the amount of work owned by each processor; communication costs are not considered. An additional advantage of the chain partitioner over the two bisection methods is that it can be used with any number of processors, whereas the bisection methods require 2^N processors. Figures 2, 3, and 4 show problem domains partitioned using RCB, RIB, and chain partitioning, respectively. When any of these repartitioning methods are used, both cell-based and molecule-based data must be remapped as well. The same lightweight communication schedules and datatransfer procedures discussed earlier can be used to perform this remapping.

For dynamic problems such as DSMC, the workload distribution can change drastically during execution, leading to a high degree of load imbalance among the processors in use. Thus, the partitioning methods described above are reapplied at fixed or varying intervals. It has been shown that, for many problems solved using a load-balancing algorithm, remapping the domain at fixed intervals can lead to poor performance. Therefore, it is desirable to either determine the optimum interval for remapping, or employ a monitoring policy which actively decides when remapping is necessary. The former choice is not practicable for most problems. Thus, in this study, a variable-interval remapping policy is investigated as well; the method employed is the Stop at Rise (SAR) policy described by Nicol and Saltz.[16] This remapping policy chooses to repartition the domain based on the value of a system degradation function *W*, which is defined as follows:

$$W(n) = \frac{\sum_{j=1}^{n} \left(t_{\max}(j) - t_{\mathrm{avg}}(j) \right) + C}{n} \qquad (1)$$

In the above, n is the number of time steps since the last remapping (which occurred in the step just before step “1”), t_{max} is the maximum amount of time required by any one processor during the jth time step (and thus the amount of time required to complete the jth time step), t_{avg} is the average time required by a processor to complete the jth time step, and C is the amount of time required to complete the remapping operation. This quantity is monitored during the computation, and represents the average processor idle time per step achieved by remapping immediately. Repartitioning is performed after the first value of n such that $W(n) > W(n-1)$ --that is, when the first local minimum is detected. The function $W(n)$ initially tends to decrease as n increases, because the remapping cost C is amortized over an increasing number of time steps. However, as n increases, the summation term in Equation (1) will eventually increase as well, indicating a loss of workload balance and a need to remap. This remapping method is advantageous in that no prior knowledge of the problem is necessary for the determination of the remapping interval, and the remapping interval can be expected to adapt to the dynamics of the problem.

These procedures were implemented into the modified DSMC3 code of Bird and ported to the 72-node Intel Paragon recently brought on line at NASA Langley Research Center. The results presented herein were obtained using up to 64 nodes on this machine.

Results and Discussion

Parallel Performance Results

It was desired to evaluate the effect of different partitioning methods on the performance of the parallel algorithm. To this end, a simplified problem (zero-thickness flat plate with a smaller domain and $48 \times 16 \times 2$ grid), with the same freestream conditions as the CNRS experiment, was solved using the parallel code set up in a variety of configurations. The test case was also run on a scalar machine, and excellent agreement was found between the flowfield and surface results produced by the two codes. Four suites of performance tests were conducted. In the first, the three partitioning methods described earlier were compared for a fixed remapping interval of 20 timesteps. These results are shown in Figure 5. It may be seen that RCB offers the poorest performance of the three methods, and that for this method the execution time actually increases as we go from 32 to 64 nodes. Additionally, it is clear that the chain partitioner yields the best performance of the three when applied to this particular problem.

Figures 6 and 7 compare the possibilities available for the remapping interval: no remapping (static partition), fixed-interval remapping, and SAR. The results in Figure 6 are for recursive inertial bisection, and those in Figure 7 were obtained using the chain partitioner. For this test case, the static partition gives very good results, particularly for large numbers of processors. As far as the other two cases are concerned, we see that the fixed-interval remapping outperforms SAR in most cases, but that the two curves follow the same trends. The exception is the crossover in Figure 6; for 64 nodes and RIB, SAR does outperform fixed-interval remapping. The results in these two figures indicate that the test case is simple enough that the cost of *any* remapping whatsoever is greater than the load imbalance suffered through retaining a static partition. Thus, it is apparent that the best remapping policy is problem-dependent.

In Figure 8, we see the performance results obtained for two different measures of workload--compute time per cell and number of molecules per cell. For this problem, we note that the two measures give very similar behavior, and it is difficult to say which method is better.

The performance results discussed here show that, in general, the benefit realized from increasing the number of processors decreases as the number of processors increases. Nevertheless, for the correct selection of partitioning and remapping methods, an appreciable decrease in run time can still be realized by running on a very large number of nodes. As a basis for comparison, the scalar version of this simulation required 930 s to run on a Sun SPARCstation 2 workstation.

CNRS Comparison Results

Figure 9 shows results for the heat-transfer rate along the top of the finite-thickness CNRS plate using the original, scalar version of the modified DSMC3 code and a $52 \times 24 \times 26$ grid. Also shown are the CNRS DSMC results and the test data itself. Note that the current results shown here follow about the same trends as the previous DSMC results, although the agreement is poor near the plate leading edge. Moreover, both sets of computed results overpredict the heating rate, particularly near the leading edge.

In order to obtain improved flowfield resolution without a substantial computational penalty, the original code was modified to sample the simulated particles on a subcell level instead of at the cell level. Figure 10 shows the heat-flux results for this new method, with four subcells per cell in the *y* direction. We see that the agreement between these results and the experimental data is better than that between the test results and the previous DSMC results; this improvement may be traced to improved grid resolution. However, the discrepancy between the experimental and computed results is still quite large at the leading edge.

Prior to conducting a grid resolution study using the parallel code, it was desired to determine which code setup would be best for the CNRS problem. Therefore, an abbreviated set of performance-evaluation runs was conducted using a moderately large ($52 \times 60 \times 26$) grid and about 860,000 molecules. The results of these tests are shown in Table 2; in each case, the code was run 1000 timesteps on 32 nodes. Note that the combination of chain partitioning and SAR yielded a slightly higher execution time than the combination of chain partitioning and fixed-interval remapping; however, the degree of load imbalance is slightly lower for the chain/SAR case. (The load imbalance is defined as the ratio of the compute time required for the bottleneck processor to the average compute time for all the nodes.) Furthermore, it is clear that the third and fourth setups are poor candidates for use in higher-resolution studies, since both require considerably larger amounts of CPU time and exhibit large degrees of load imbalance. Based on these results, the code was configured to utilize the chain partitioner in conjunction with the SAR remapping policy, since SAR appeared to run only slightly slower for the case considered and the fixed remapping interval used here (once every 50 timesteps) may not be the best remapping interval for all the cases considered.

In order to find a grid-independent solution, the parallel code was next used to obtain results for successively finer grids; the results for the final grid are shown in Figure 11. This case utilized a $52 \times 96 \times 26$ grid and approximately 1.4 million simulated molecules. It may be seen that the agreement between the DSMC results and experimental data is now much better, although the agreement is still not very good at the leading edge. We may also compare the surface pressure results, as shown in Figure 12. It is interesting to note that, while the DSMC heat-flux results compare well to the experimental data, the same cannot be said for the surface-pressure results. In fact, it may be seen that the computed results do not even follow the same trend as the experimental results.

It should be reiterated that the problem considered here required very simple flow physics; moreover, the geometry itself was quite simple. One may ask whether the present method could be readily expanded to more comprehensive flow physics and more complex surface geometries. As far as the latter is concerned, inclusion of other physical phenomena should be relatively straightforward. In DSMC, any physical process such as dissociation or ionization requires a collision between simulated particles, and the collision coding used herein is virtually unchanged from the scalar algorithm. The main difference would be that new arrays would be necessary to keep track of the additional particle information (such as vibrational energy state), and these arrays would have to be distributed in the same fashion as other data arrays. In order to incorporate more general geometries into the code capability, a nonuniform grid would most likely be necessary. Such a grid would necessarily complicate the movement and indexing phases of the method, and these added difficulties would translate into further difficulties with respect to parallelization. However, these problems can most likely be overcome, and will be part of the focus of further work.

An additional note regarding use of the parallel code on more complex configurations is in order. As mentioned before, the best choice of partitioning strategy is problem-dependent, and methods which worked well for the simple geometries considered herein may produce poor performance for other geometries. For instance, the chain partitioner discussed herein was quite useful for both the test case and the CNRS case. However, it may not be as useful in flows where a large percentage of the particles move in a direction other than the freestream flow direction, such as blunt-body wake flows. In any case, one should evaluate the available domain-decomposition options for a new problem prior to attempting a full-blown simulation of the problem.

Conclusions

The parallel implementation of the DSMC method presented here was successful in producing results which compared well with scalar DSMC results for the simple test case. Through use of the parallel code, it was possible to increase grid resolution and still obtain solutions for the CNRS comparison case in a reasonable amount of time. The grid-independent results obtained using the parallel code showed good agreement with experimental heat-transfer data; however, the computed pressure distribution still compared very poorly with the measured distribution.

Performance results for the test case and the CNRS case indicate that the best partitioning and remapping policies are problem dependent; some of the load-balancing strategies which produced acceptable performance for the test case worked poorly for the CNRS case. Thus, it is important to know something about the flow under consideration before deciding what set of parallel-execution parameters to employ.

Acknowledgments

This work is supported in part by NASA Cooperative Agreement NCCI-112, the Mars Mission Research Center funded by NASA Grant NAGW-1331, a National Defense Science and Engineering Graduate Fellowship, NASA Grant NAG-1-1560, ARPA/NASA Grant NAG-1-1485, and NSF/NASA Grant ASC 9213821. Computer resources were provided by NASA Langley Research Center.

References

1. Bird, G.A., “Monte Carlo Simulation in an Engineering Context”, *Progress in Aeronautics and Astronautics: Rarefied Gas Dynamics*, Vol. 74, Pt. 1, ed. Sam S. Fisher, AIAA, New York, 1981, pp. 239-255.
2. Rault, D. F. G., “Aerodynamics of Shuttle Orbiter at High Altitudes”, AIAA Paper 93-2815, July 1993.
3. Woronowicz, M. S. and D. F. G. Rault, “On Predicting Contamination Levels of HALOE Optics aboard UARS Using Direct Simulation Monte Carlo”, AIAA Paper 93-2869, July 1993.
4. Taylor, J. C., A. B. Carlson, and H. A. Hassan, “Monte Carlo Simulation of Radiating Reentry Flows”, AIAA Paper 93-2809, July 1993.
5. Wilmoth, R. G., “Adaptive Domain Decomposition for Monte Carlo Simulations on Parallel Processors”, in *Proceedings of the 17th International Symposium on Rarefied Gas Dynamics*, ed. A. E. Beylich, VCH Publishers, New York, 1991, pp. 709-716.
6. Allegre, J., M. Raffin, A. Chpoun, and L. Gottesdiener, “Rarefied Hypersonic Flow over a Flat Plate with Truncated Leading Edge”, presented at the 18th International Symposium on Rarefied Gas Dynamics, Vancouver, July 1992.
7. Hash, D. B., J. N. Moss, and H. A. Hassan, “Direct Simulation of Diatomic Gases Using the Generalized Hard Sphere Model”, AIAA Paper 93-0730, January 1993.
8. Rault, D. F. G., “Towards an Efficient Three-Dimensional DSMC Code for Complex Geometry Problems”, presented at the 18th International Symposium on Rarefied Gas Dynamics, Vancouver, July 1992.
9. Bird, G. A., *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, Clarendon Press, Oxford, 1994.
10. Borgnakke, C., and Larsen, P. S., “Statistical Collision Model for Monte Carlo Simulation of Polyatomic Gas Mixtures”, *Journal of Computational Physics*, Vol. 18, No. 3, 1975, pp. 405-420.
11. Das, R., and J. Saltz, “Parallelizing Molecular Dynamics Codes Using the Parti Software”, in *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, March 1993, pp. 187-192.
12. Moon, B., and J. Saltz, “Adaptive Runtime Support for Direct Simulation Monte Carlo Methods on Distributed Memory Architectures”, in *Proceedings of the Scalable High Performance Computing Conference (SHPCC94)*, IEEE Computer Society Press, Knoxville, TN, May 1994, pp. 176-183.
13. Berger, M. J., and S. H. Bokhari, “A Partitioning Strategy for Nonuniform Problems on Multiprocessors”, *IEEE Transactions on Computers*, Vol. 36, No. 5, May 1987, pp. 570-580.
14. Nour-Omid, B., A. Raefsky, and G. Lyzenga, “Solving Finite Element Equations on Concurrent Computers”, in *Proceedings of Symposium on Parallel Computations and their Impact on Mechanics*, Boston, December 1987.
15. Berryman, H., J. Saltz, and J. Scroggs, “Execution Time Support for Adaptive Scientific Algorithms on Distributed Memory Machines”, *Concurrency: Practice and Experience*, Vol. 3, No. 3, June 1991, p.159-178.
16. Nicol, D. M. and J. Saltz, “Dynamic Remapping of Parallel Computations with Varying Resource Demands”, *IEEE Transactions on Computers*, Vol. 37, No.9, September 1988, pp.1073-1087.

Tables

Table 1. CNRS test conditions.

Property	Value
M_∞	20
T_∞	14 K
Re	8380
p_0	10 bar
T_0	1100 K
T_w	290 K

Table 2. CNRS case performance results.

Partition/ Remap interval	Execution time (seconds)	Degree of load imbalance
Chain/SAR	1217.3	1.019
Chain/Fixed	1198.8	1.020
RIB/SAR	1381.4	1.065
Static/--	2015.4	1.775

Figures

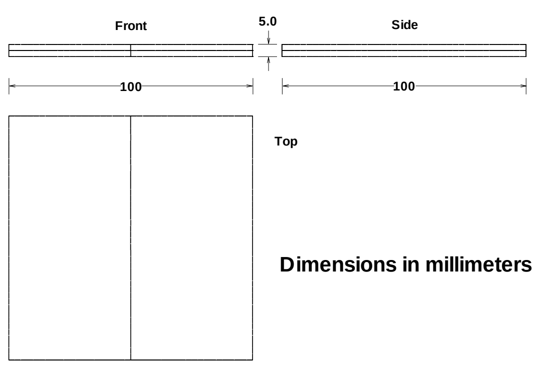

Figure 1. CNRS flat-plate geometry.

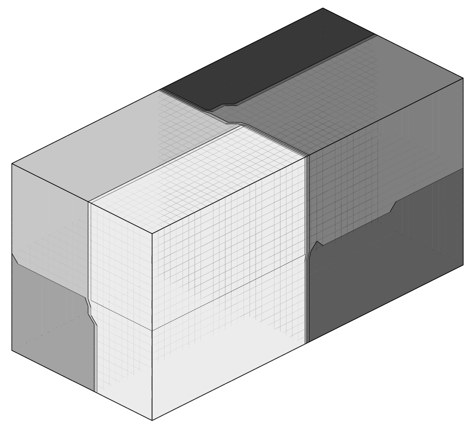

Figure 2. Domain decomposition with RCB.

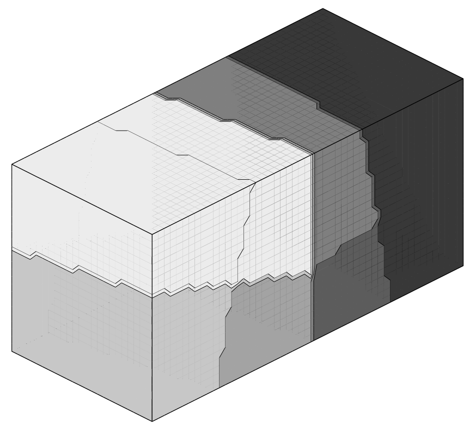

Figure 3. Domain decomposition with RIB.

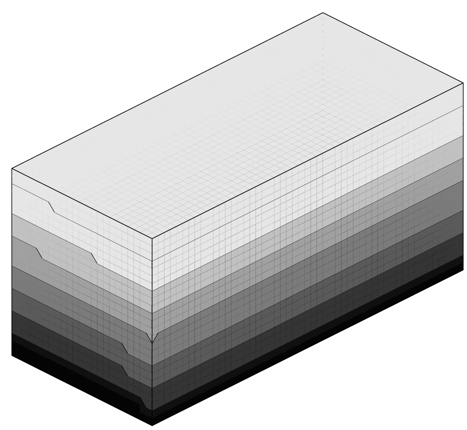

Figure 4. Domain decomposition with the chain partitioner.

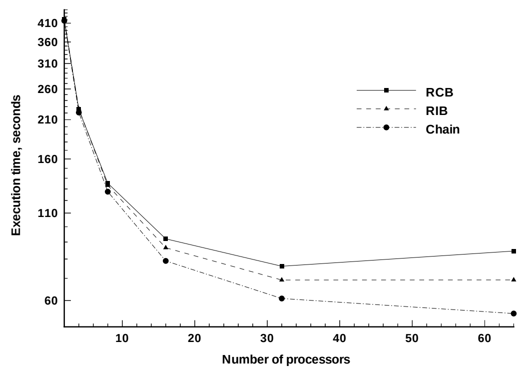

Figure 5. Performance results: Effect of partition method.

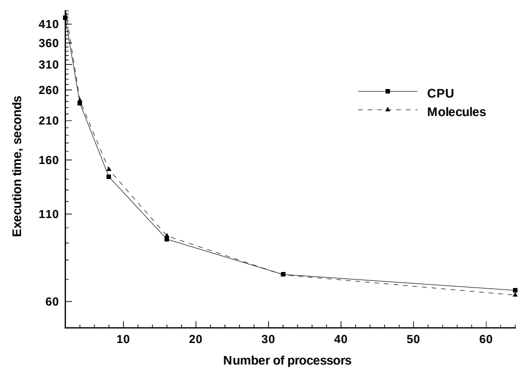

Figure 8. Performance results: Effect of workload measure.

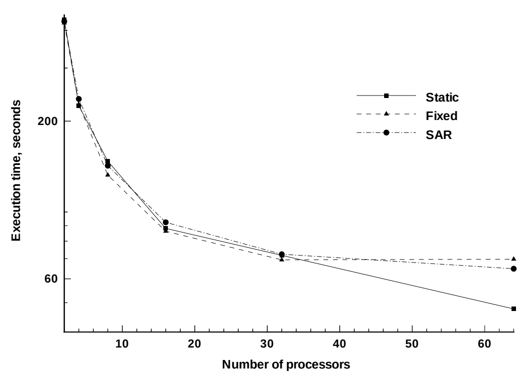

Figure 6. Performance results: Effect of remapping method (RIB).

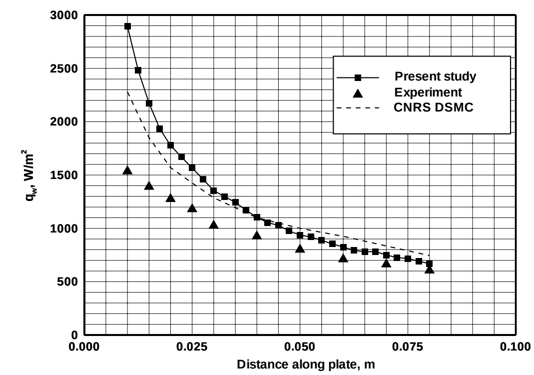

Figure 9. Heat flux results at $z = 0$ for the scalar code ($52 \times 24 \times 26$ grid,1 subcell per cell).

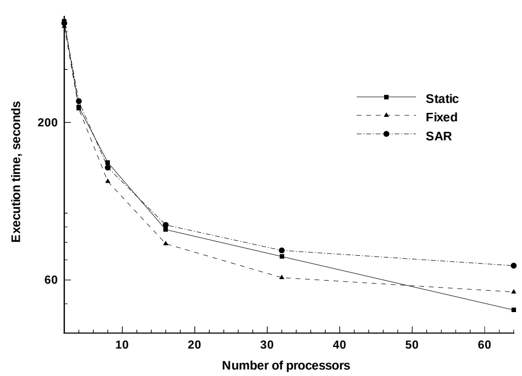

Figure 7. Performance results: Effect of remapping method (chain partitioner).

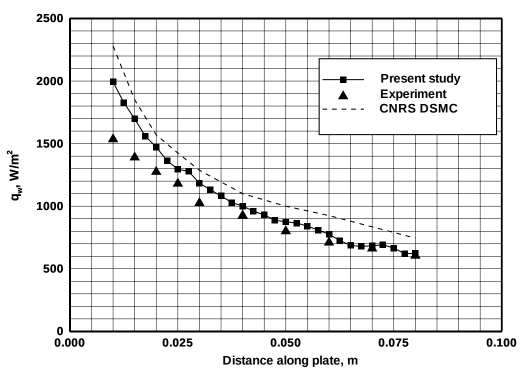

Figure 10. Heat flux results at $z = 0$ for the scalar code ($52 \times 24 \times 26$ grid, 4 subcells per cell in the y direction, flow sampled on a per-subcell basis).

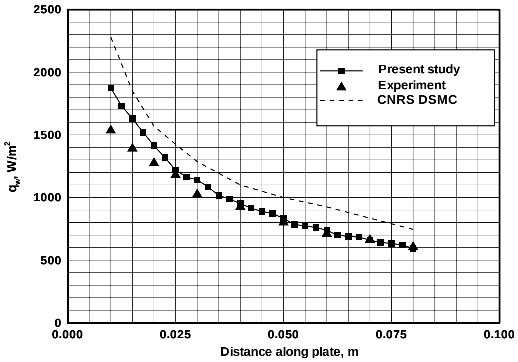

Figure 11. Heat flux results at $z = 0$ for the parallel code ($52 \times 96 \times 26$ grid, 1 subcell per cell).

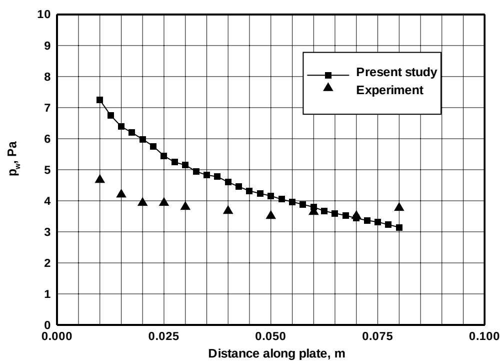

Figure 12. Surface-pressure results at $z = 0$ for the parallel code ($52 \times 96 \times 26$ grid, 1 subcell per cell).